U.S. DEPARTMENT OF COMMERCE
National Institute of Standards and Technology

Computer Systems Laboratory

NISTIR 4630

Performance Evaluation of Hypercube Applications: Using a Global Clock and Time Dilation

Robert D. Snelick

U.S. DEPARTMENT OF COMMERCE
National Institute of Standards and Technology
Computer Systems Laboratory
Advanced Systems Division
Gaithersburg, MD 20899

July 1991

CMRF

COMPUTER MEASUREMENT
RESEARCH FACILITY
FOR HIGH PERFORMANCE
PARALLEL COMPUTATION

Partially sponsored by
- Defense Advanced Research Projects Agency
- Department of Energy

NISTIR 4630

Performance Evaluation of Hypercube Applications: Using a Global Clock and Time Dilation

Robert D. Snelick

Advanced Systems Division
Computer Systems Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899

Partially sponsored by
- Defense Advanced Research Projects Agency
- Department of Energy

U.S. Department of Commerce, Robert A. Mosbacher, Secretary

National Institute of Standards and Technology
John W. Lyons, Director

July 1991

TABLE OF CONTENTS

Page

Performance Evaluation of Hypercube Applications: Using a Global Clock and Time Dilation

Robert D. Snelick

The speed of communication is very important in the performance of programs for loosely-coupled machines. A precursory study introduced an emulation technique called time dilation [1] that investigated the effects of communication speeds by varying the ratio of two parameters: communication time and computation time. The analysis of this technique was based on local measurements of a node's system resource utilization. A problem for loosely-coupled systems with local clocks is that observations cannot directly resolve communication delays into logical and physical transport contributions. This work separates the communication component of a program (via a global clock) into two states: logical and physical delays. True measurements calibrate the indirect dilation method for a sharper, quantitative interpretation, that it does not otherwise have.

Time dilation, coupled with low perturbing global measurement, provides an environment that offers insight for the development and analysis of concurrent algorithms and architectures. Measurement of communication service states indicates categorically sources of delay in the communication structure. The observables are applied to evaluate a set of example hypercube applications. Since the analysis is based upon system resources, which are invariant across all hypercube programs, it forms a reliable base for comparison. Results demonstrate that improvement to infrequent system states can cause significant changes in program behavior. This is most apparent for synchronous algorithms.

Key words: communication; emulation; global clock; hypercube; measurements; performance; resource usage; service states; time dilation.

 Partially sponsored by the Defense Advanced Research Projects Agency, 1400 Wilson Boulevard, Arlington, Virginia 22209 under ARPA Order No. 7223, April 15, 1987.

1. Introduction

Performance evaluation of loosely-coupled multiprocessors is critical for exploring parallel program characteristics and parallel architecture behavior. In loosely-coupled multiprocessors, synchronization and data sharing are achieved by explicit message passing. Therefore, the speed of communication is very important in the overall performance of such machines. Current methods of investigating the effects of communication speeds are modeling and simulation. We can evaluate the performance of hypercube applications by integrating measurement procedures and a performance evaluation technique that were inspired by a hybrid measurement system [5]. A precursory study introduced a technique called *time dilation* [1] that investigated the effects of communication speeds by varying the ratio of two parameters: communication time and computation time. This technique was based on local measurements of a node's system resource utilization. A problem for loosely-coupled systems with local clocks is that observations cannot directly resolve logical and physical transport contributions for communication delays. This left the interpretation and analysis of the technique clouded with obscurities. This work separates the communication component of a program (via a global clock) into two elements: logical and physical delays. True measurements give the indirect dilation method a sharper, quantitative interpretation. In addition, quantifying sources of performance delay provides the programmer insight on the efficiency of his algorithm and highlights opportunities for improvement. This information is unavailable with local time.

The time dilation experiments and event tracing were performed on our specially instrumented iPSC/1. Four primary types of performance data, referred to as system service states, are captured to provide the information needed to analyze applications. As mentioned, the communication component of the program is separated into two states: logical and physical transport delays. Other service states measured to aid in the evaluation of hypercube applications are the time for user computation and overhead. Overhead is the time for all remaining activities excluding logical delays, physical delays, and user computation.

The test programs are three synthetic benchmarks that outline several broad application modes by communication dependencies. The benchmarks provide a parameter set that allows the user to adjust the computation-communication balance of the algorithm. This allows testing across a spectrum of communication dependencies and computation-communication granularities. Six example applications are presented that demonstrate the virtue of this technique. The benchmarks are not essential for this study--they are used to illustrate effects. The main emphasis of this study is the test method.

Our measurement approach accurately characterizes the system-level resource usage of a particular application. These characterizations simplify the task of evaluating hypercube applications. The information is important in quickly uncovering potential bottlenecks, as well as highlighting good algorithmic features. The tests also reveal salient features of the architecture (such as message retransmissions) that do not lend themselves easily to modeling or simulation. Coupled with the dilation experiments, this system service measurement approach provides an environment which will offer insight

into the development and analysis of concurrent algorithms and architectures. Since the analysis is based upon system resources, which are invariant across all hypercube programs, it forms a reliable base for comparison [4]. Thus the method is suitable to all hypercube applications.

2. Time Dilation Background

Computation and communication form a natural dichotomy on loosely-coupled multiprocessor resources, since with each node the processor has to handle both duties. Modeling and simulation are common techniques used to estimate machine performance based on the speed of these resources. The accuracy of the resulting performance estimates is often questionable, since such techniques cannot usually take into account all of the system detail--they become intractable in effort. Any real, hardware implementation of a multicomputer immediately fixes the speed of its resources, and can only yield a single point on the performance curve. An emulation technique, time dilation, *appears* to make physical transport faster through *slowed* computation. Time dilation provides an accurate method of investigating the performance of a given program on a variety of the physical transport speeds of a real machine. Time dilation is implemented by allowing the user to increase, or dilate, the computational portion of the code by a specified factor, thereby creating the appearance that the communication speed has increased in relationship to the rate of computation.

To perform time dilation experiments, the operating system of the machine under investigation must be instrumented with code to perform the dilation function. This additional operating system code relies on a high-speed clock to measure accurately and quickly the computation and communication intervals. The high speed clock is provided by our Loosely-coupled TRAce Measurement System (LTRAMS). LTRAMS is a hybrid performance measurement tool that, among other things, provides a readable clock accurate to one microsecond [5]. The implementation of time dilation is described in [1].

To utilize time dilation the user first runs an application with a dilation factor of one (no change), and then runs the application with other dilation factors, all greater than one. The overall time for the application to complete each run is "normalized" by dividing the time obtained from the run by the dilation factor for that particular run. The difference in normalized times gives an indication to what extent a faster transport system would benefit a particular application.

2.1 Previous Results

In a previous study, local clocks were used to obtain performance data for a hypercube application tested with time dilation. The clocks captured the node's local service demand components, which were separated into four partitions: receive time, send time, computation time, and communication interrupt time. With this framework, time dilation's evaluation offered some insight on performance, but could not directly resolve logical and physical transport contributions for communication delays. Resource utilization components, organized in this way, lacked detail. For example, most of the performance acceleration originated from the receive time component, but its constituents could not be distinguished. The emphasis for the remainder of this text is a more detailed measurement of communication delays (as well as other system service states) and the role they play in characterizing hypercube applications. This work also extends the set of applications tested with dilation.

3. A Communication Benchmark Set

This section describes three synthetic communication benchmarks used in the experiments to follow. The programs delineate several broad application modes by communication dependencies [2], [3]. The benchmarks provide a parameter set that can be used to adjust the computation-communication balance of the algorithm.

3.1 High Interdependencies: A Ring Model

The synthetic ring benchmark models applications which exemplify global process dependencies. Molecular dynamics codes have computations of this character. The communication patterns for the Ring are given in Figure 1. Data message flow in the ring is uni-directional.

The program works as follows: A synthetic ring with nearest neighbor connections is created with N logical nodes. Each node (process) will originate a given number of messages, and additionally, process all other messages passing by. Each message makes a complete circuit around the ring, while being processed synthetically by each node. Once a message returns to its originator, it is removed from the ring network. Another message will be sent until that node has sent all of its messages. When all nodes have processed all of their messages the program is complete.

The ring provides a set of parameters that allows the user to simulate a wide range of application settings. The list of parameters includes the number of nodes in the ring, message length, the number of messages, and computation per datum. Communication in the ring is semi-synchronous, with messages being acknowledged within the program on a one-to-one basis. This flow control limits the number of messages piling up on a given node, thus preventing buffer overflows. For this algorithm all communications (*i.e., messages*) are between physically neighboring nodes on the hypercube.

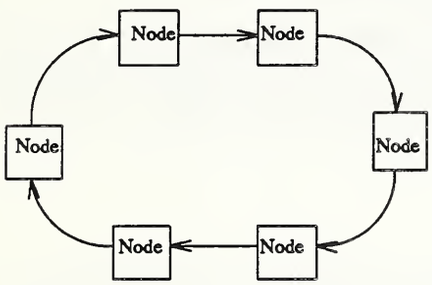

Figure 1. Ring Communication Patterns.

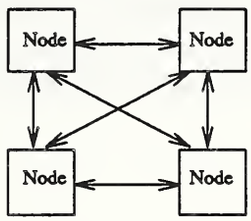

Figure 2. Random Communication Patterns.

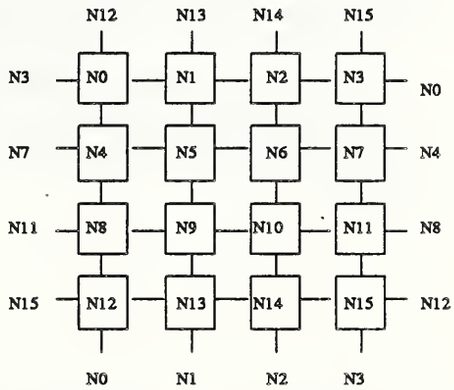

Figure 3. Mesh Communication Patterns.

3.2 Low Dependencies: Random Communication (RC) Model

Another synthetic benchmark for process communication depicts computational objects (nodes) that can be scheduled nearly independently (hereafter, RC, for Random Communication). This circumstance might be radiation transport, or within a computer system, storage scavenging on linked lists. It represents the other end of the interdependency spectrum from the ring. The algorithm implements a pool of n logical nodes (processes), from each of which random messages are initiated, dispatched, received and processed. The communication patterns for this algorithm are shown in Figure 2. Each node in the graph has n-1 edges, each of equal weight in the logical structure.

Each node is initialized with an equal amount of data, hence the workload is initially balanced. Nodes process some workload and randomly distribute other chunks of their workload to other nodes. In general the algorithm works as follows: A node begins and continues processing its data until it is notified within its work set (via a special value) to *pass* some of it on or until it finishes that set of data. When it must yield part of its present work set, it randomly chooses (based upon data values) a destination node and determines how much to dispatch. While the node is processing its workload it will periodically check its message queue for more work. If there are pending message(s) the

node receives the message(s) and schedules its added workload. Note, that as long as a node has work to do, it can continue processing. It is not dependent on messages from other nodes (i.e., the algorithm is largely asynchronous). However, it periodically checks its queues to prevent buffer overflow. Flow control is handled by limiting the number of messages a node can send out before it receives an acknowledgement of those messages. Whenever the node completes all of its current assigned data, it initiates a termination message. It then sits idle awaiting a message, which can be a data message (*new work*) or a termination message. Once all nodes have communicated that the entire workload has been processed, results are recorded and the pool of nodes is dissolved.

Parameters for this benchmark include the number of nodes, message length, computation per datum, and frequency of message originations (work dispatching). The latter two parameters are used to adjust the computation-communication granularity of the algorithm.

3.3 Local Dependencies: A Mesh Model

A middle ground in the interdependency spectrum has models in which the next state is a function of a small number of neighboring processes. The model chosen to represent this has a two-dimensional mesh structure with nearest neighbor communication dependencies (Figure 3). A description of the algorithm follows.

An N by M mesh with toroidal (wrap-around) connections is created with N*M logical nodes. Each node in the mesh communicates with four nearest neighbors and has a well-defined structure with a North-South-East-West pattern of communication. In the test examples to follow, N=M.

The program iterates state-by-state, and each state has two distinct phases, communication and computation. The communication phase does information exchanges and node synchronization. The computation phase defines the amount of computation performed by each node (process) between synchronizations. The mesh program stops after performing a given number of state iterations. Communication places a global constraint on the overall computations. A node (process) needs values obtained from the previous states of its nearest neighbors to continue; this fact prevents it from outdistancing its neighbors. The algorithm proceeds in a lock-step (synchronous) fashion. The distinction between a synchronous and asynchronous algorithm can be an important application attribute in analyzing results for time dilation experiments.

Parameters for the mesh include the number of computation-communication steps, message length, and computation calculations per datum. For these experiments, the mesh is mapped perfectly so that its logical structure matches the underlying physical structure (i.e., nearest neighbor communication is pre-dominant at the physical level).

4. Analysis and Interpretation

4.1 Elements of Performance Variation

Programs on an iPSC-1 hypercube can use either asynchronous or synchronous communication protocols (not to be confused with the classification of an algorithm). Our test set uses the latter. The synchronous protocol has two latency components, *logical delay* and *physical delay*. **Logical delay** is defined as the time the destination node initiates a receive request until the source node issues the send command. This does not include the time it takes for the message transfer nor any other delays. If the source node issues a send command before the destination node initiates a receive request, there is no logical delay. Logical delays can indicate algorithmic bottlenecks. **Physical delay** is the time a message takes to travel over the physical medium from sender node to recipient. It does not include other delays such as packet formation, buffer allocation, queuing delays at other nodes, etc. This distinction does not change for multiple packet messages. The packet is the element of measurement, not the message. So, multiple packet messages and multiple hop messages are reduced to the packet level, and just the time for each packet is accumulated. Physical delay includes the time for user level packets as well as acknowledgement packets initiated by the system. Together, logical and physical transport latencies indicate whether a poorly performing application needs a new algorithm (less logical delay) or faster interconnection hardware (less physical delay). Other service state data measured to aid in the evaluation of hypercube applications are the time for user computation and overhead. **User computation** is composed of application computation and operating system services (not associated with communication). It does not include the time for interrupts to handle communication tasks. **Overhead** is the time for all remaining activities not included in logical delay, physical delay, and user computation. Overhead includes such activities as formation of packet headers, buffer allocation, communication system interrupts, and time loss due to message retransmissions. The last element, message retransmission, can be an important source of performance variation for this particular architecture.

Two scenarios of message retransmission exist. If a failure (which causes a retransmission) occurs and the receiving process queue is long enough such that the sequel retransmission is received before the receiving process queue is exhausted, the retransmission has minimal effect on algorithm (synchronization) performance. If, on the other hand, the receiving process queue is exhausted and the process is forced to wait for the retransmitted message, synchronization is skewed. Performance degradation can be substantial from frequent message retransmissions. When the receiving process is subject to a retransmission it may be delayed by a factor of 100 or more for the reception of that message. Such delay can have adverse effects on the performance of an application. This effect on the communication structure, algorithmically and architecturally is easy for us to measure, but is difficult to account for in techniques such as modeling and simulation. Salient features such as this should be apparent to programmers of such architectures.

4.2 Method

The implementation and measurement of time dilation were performed on our specially instrumented 16 processor iPSC-1 hypercube system. The measurement system provides time tracings, as well as, a readable clock accurate to one microsecond. Our performance analysis is based on event driven measurements. Each processor in the system has an instrumentation board that captures local trace information. Global timing is achieved by using a global reset to synchronize all counters. Each measurement board gets its tick from a common clock, thus preventing drift. Data then can be paired across nodes to obtain a global sense of timing. Direct global timings are more *expensive* than local timings (i.e., in terms of the complexity of obtaining measurements and space requirements). Local timings can be simple and *inexpensive*, for example, a set of accumulator registers to capture average resource utilization. Local timing is very useful in obtaining performance data [4], but cannot resolve performance variation in a global domain. Global timing can provide performance detail unobtainable through local timing. A global event trace can be used to reconstruct the state of the entire system. Our performance measurement procedure relies on global timings. The time dilation technique uses local clocks.

Logical and physical delay state information is first traced locally (via operating system events) at each node and then matched up across nodes to obtain the interval time. User computation is obtained locally at each node by direct measurement. Overhead is calculated by subtracting these elements from overall execution time of the node. Service state time is the average time for N nodes (the test programs are homogeneous). The number of retransmissions are obtained by counting special operating system events which designate when a message is retransmitted.

Other elements of performance variation for overhead (such as interrupts for handling communication tasks and other communication activities) can be further subdivided using the measurement system, but their resource consumption is of little consequence in the evaluation of hypercube applications. Thus, they are grouped into overhead. The benefit gained in obtaining a finer detailed event-trace did not justify the expense. This set of performance parameters is an adequate basis to study sources of performance variation.

4.3 Analysis of Time Dilation

To evaluate results of time dilation, a normalized time (T_D) is defined as the average node service time (T) for a component (e.g., logical delay equals the average logical delay for N nodes) divided by the dilation factor (D). The dilation factor (D) is equivalent to the emulated speedup of the physical transport system. E is the overall execution time of the dilated program and E_D is the normalized program time.

$$T_D = \frac{T}{D} \quad \text{and} \quad E_D = \frac{E}{D}$$

In theory, normalized time gives a measure of what a component's time, such as logical delay, or the overall execution time of the program will be on a machine with a faster physical transport system. T_D is a generic term for our system service states: logical delay (L_D), physical delay (P_D), user computation (U_D), and overhead (O_D). Application performance is summarized in the following equation:

$$E_D = L_D + P_D + U_D + O_D$$

Another measure that is useful is the speedup of the application for a given D. This is obtained by dividing E_1 (E_1 is the overall execution time of an undilated program) by E_D with a D greater than one.

$$Speedup = \frac{E_1}{E_D} \quad (where\ D > 1)$$

5. Results

To illustrate results, each benchmark is tested with two parameter settings. One parameter setting investigates communication intensive applications while the other investigates computation intensive applications. The relative grain of computation to communication along with the program system service occupancy is given for each experiment in table I. The computation-communication grain size gives the number of data items processed (integer multiplication in these examples) between communication interaction points. User computation, logical delay, physical delay, and overhead are given as percentages of overall program time. These percentages are recorded from undilated programs. A prefix of CM (communication intensive) or CP (computation intensive) is attached to each algorithm to aid explanation in comparing the experiments. In addition, a suffix is attached to distinguish a given dilation factor for the test. For example, CM-RC-2 refers to an experiment for the random communication benchmark with a low computation-communication grain size, ran with a dilation factor of 2.

Test Program	Comp-Comm Grain	User Computation	Logical Delay	Physical Delay	Overhead Delay
CM-Ring	128	17.1%	51.8%	4.8%	26.3%
CM-Mesh	640	22.0%	36.0%	3.4%	38.6%
CM-RC	383	87.1%	6.5%	0.6%	5.8%
CP-Ring	12,800	88.0%	7.3%	0.9%	3.8%
CP-Mesh	102,400	94.7%	0.7%	1.0%	3.6%
CP-RC	17,430	51.5%	48.4%	0.0%	0.1%

Table I. Computation-Communication Grain Size and Component Demands.

Tables A1 through A6, in Appendix A, show the performance data for the aforementioned test set. The tables present data for each of the programs with dilation factors ranging from one to sixteen. The data shown are for one trial run and are typical instances of other similar test results. That is, the variation from one run to another is minimal. The first column E_D gives the normalized elapsed time for the program. The next four columns indicate normalized time for user computation (U_D), logical delay (L_D), physical delay (P_D), and overhead (O_D) state occupancies. Other observations that give insight into the analysis are also included in the tables. %LM is the percent of messages that had some logical delay (i.e., % of messages with logical delay > 0). The average logical delay per logical message (A_D) is given next. It gives an indication of how long, on average, a node waits for a message, when it is required to wait (zero entries are discarded) for a message. A_D is a normalized statistic. Max Log gives the maximum delay a node waited for any one message. The numbers for Max Log are actual measurements; that is, they are not normalized. TM is the total number of user messages for the experiment. The number of retries is a sum of message retransmissions that occur for all nodes.

CM-Ring and CM-Mesh showed significant performance improvement with the faster transport system ($D > 1$ experiments). These programs are very sensitive to ΔP_D even though P_D is small. The sensitivity slope of E_D versus P_D indicate that a minor improvement to a state occupancy (P_D) can alter program behavior significantly. Figure 4 illustrates the slopes (denoted as m_1 and m_2) for E_D versus P_D for CM-Ring and CM-Mesh for the dilation interval of 1 and 2.

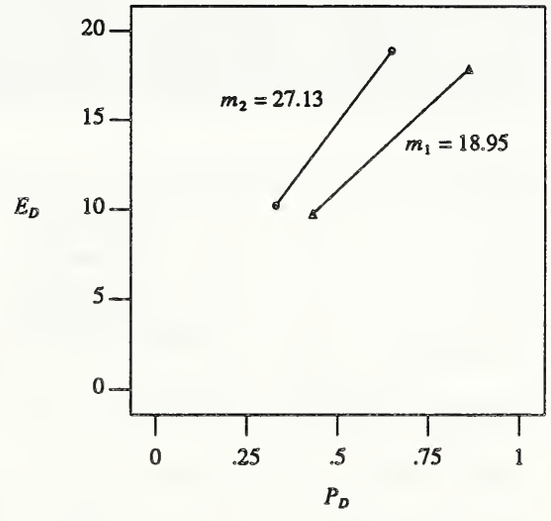

Figure 4. Slopes (m_1, m_2) of E_D vs. P_D for CM-Ring and CM-Mesh.

As shown, the impact can be quite dramatic; slopes of 18.95 and 27.13 are observed for CM-Ring and CM-Mesh, respectively. Hence, modification of state occupancies, regardless of their relative magnitude, can cause significant changes in program behavior. This is most apparent for synchronous algorithms (e.g., Ring and Mesh).

Another important factor that contributes to the increased performance seen with the dilation experiments is the decreased frequency of message retransmissions as a faster physical transport is emulated. CM-Ring and CM-Mesh show performance increases of 232% and 246%, respectively. The computation intensive counterparts of these applications offer little improvement with dilation. The logical characteristics of the RC benchmark limit its performance with either computation-communication granularity setting. The two regions of application settings show contrasting results. CM-RC exhibits small logical delays even though it is burdened with frequent communication. Since RC is only slightly dependent on external calculations to continue processing, it is relatively immune from the adverse effects of synchronization. With CM-RC the communication frequency is high enough so that all processors are kept busy and load balance remains. CM-RC represents a case of a communication intensive application that runs efficiently on this particular architecture. However, CP-RC experiences serious load imbalances that severely hinder its performance. Even though significant logical delay exists, a faster transport system will not help, because the logical delay is inherent into the algorithm and not in the system as seen in CM-Ring and CM-Mesh. Because of this imbalance, nodes sit idle awaiting data for long stretches of time. CP-RC is an inefficient application and is a candidate for redesign. Therefore, in both cases RC will not benefit from a faster

transport system. Next, each experiment is examined more closely and subtle features of each are pointed out.

5.0.1 CM-Ring. CM-Ring shows significant performance improvement with a faster transport system. This is evident because CM-Ring-1 spends 83% of its time doing non-user computation work. With a transport system that is twice as fast, CM-Ring-2 execution time accelerates by 46%. 67% of this comes from logical delay reduction, 5% comes from physical transport improvement, and the remaining 28% originates from the reduction in overhead delays. The reductions in logical and overhead delay are aided by the decrease in message retransmissions. Retransmissions are nearly cut in half. Figure 5 graphically illustrates results of CM-Ring for dilation factors from one to sixteen.

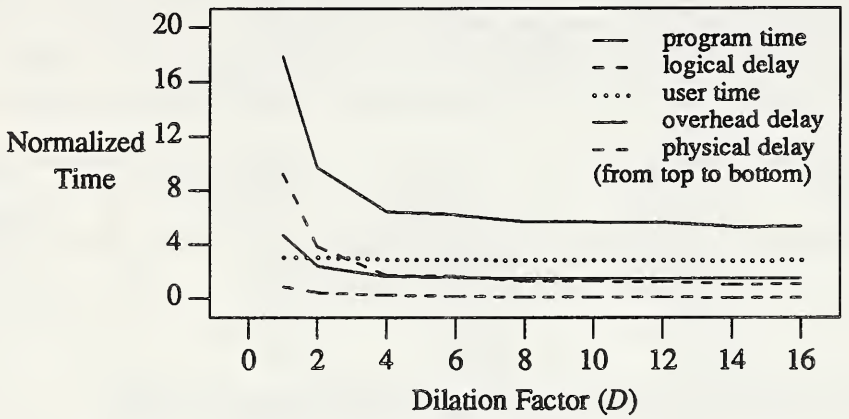

Figure 5. Normalized Component Time for CM-Ring.

Normalized times are plotted for the four system service states and overall execution time. This graph indicates where the performance acceleration originates. As clearly shown, most benefits are realized up to D equals 4, whereupon a computation-communication balance is reached and performance acceleration is no longer prominent. From CM-Ring-2 to CM-Ring-4 message retransmissions are reduced further by a factor of 4. Logical delay is reduced by another 56% (3.83 s to 1.70 s) whereas program time is accelerated by another 34%.

Note that even though it is the physical transport that is being enhanced, it directly contributes little to overall performance improvement. However, the enhanced physical transport (i.e., the dilation process) causes a reduction in the number of message retransmissions which then lessens logical and overhead delays. This relationship is exemplified in the following linear estimator (Equation 1) for E_D.

$$\hat{E}_D \approx \bar{U} + \frac{P_1}{D} + b_1(MR) + b_0 \qquad \text{Eq. (1)}$$

$$\hat{E}_D \approx 2.89 + \frac{.86}{D} + .024563(MR) + 1.5498 \qquad \text{Eq. (2)}$$

$\bar{U}$ is the mean user service time for the particular experiment and MR is the number of message retransmissions. $\frac{P_1}{D}$ is the physical transport delay for a program run with a dilation factor of one, P_1, divided by the anticipated dilation factor D. Physical transport delay in itself can be predicted by a simple linear regression model if the number of packets and the size of the packets are known. $b_1(MR) + b_0$ gives the logical and overhead delays of the program as a function of the number of message retransmissions. Equation 2 is an estimator for the CM-Ring example given in Table A1.

Another interesting point for CM-Ring is that A_D shortens with a faster transport system. For CM-Ring-1 the average wait is 16.4 ms and stablizes at just over 1 ms for higher dilation factors. So, even though there is an increase in messages with logical waits (%LM), the wait period for each is much shorter with the faster transport system. This may be an indication that synchronization is improving. The nodes are waiting more often for message originations, but have shorter wait periods. Ring is the only model where the number of messages with a logical wait increases.

5.0.2 CM-Mesh. Overall results from CM-Mesh are similar to those found in CM-Ring. Ring and Mesh are similar in that they both have synchronous communication structure. CM-Mesh-2 shows a performance acceleration of 46% over CM-Mesh-1. However, unlike CM-Ring, only 42% of the increase is seen from logical delay improvements, whereas 54% comes from other overhead sources. As with CM-Ring, a major source of performance acceleration originates from the decreased number of message retransmissions. Figure 6 shows the relationship between the number of message retransmissions and E_D, L_D, and P_D.

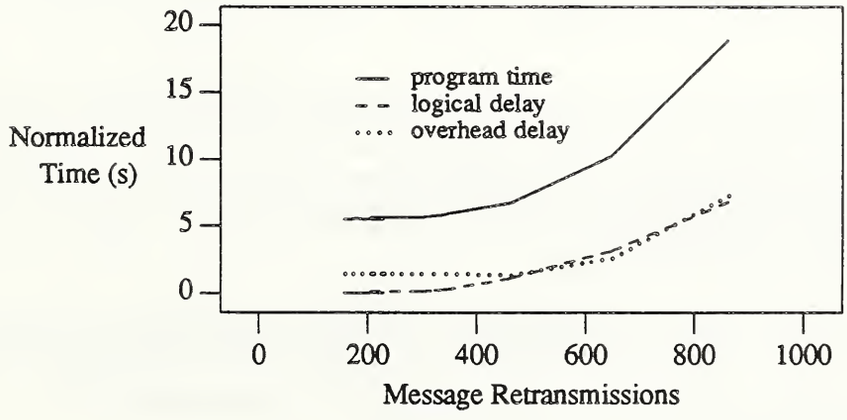

Figure 6. E_D, L_D, and O_D versus Message Retransmissions for CM-Mesh.

As shown, the resource usage of these states decrease as the number of retransmissions

become less frequent. Unlike CM-Ring, the number of messages with logical delay decreases significantly as shown in table A3 for CM-Mesh (i.e., 40% to 3%). This indicates that synchronization is greatly improved with the faster transport system.

5.0.3 CM-RC. CM-RC is a communication intensive application that did not show significant improvements with a faster transport system. Even though the computation-communication rate is low (communication intensive), this does not translate to large logical delays as seen in CM-Ring and CM-Mesh. As mentioned, in RC a node is not entirely dependent on messages from other nodes to continue processing; it is semi-asynchronous. As long as a node has work to do it can isolate itself from other nodes in the set. Only when it has completed its current workload will it be dependent on other nodes to continue. So if the workload in the network remains relatively balanced, this algorithm can run efficiently. This is an example of a good algorithm-architecture match for a communication intensive application.

CM-RC has statistical characteristics distinct from those of the other communication intensive applications studied. Since receives are algorithmically asynchronous in RC, message retransmissions are less invasive than in the Ring or Mesh. As a result, overhead for CM-RC makes up only 6% of the undilated program. Also the percentage of messages with logical delay is very small. For this algorithm, it is an indication of good load balance. That is, most of the time data was available to process. However, when CM-RC is required to wait for data, it waits for longer periods of time than CM-Ring or CM-Mesh. This reflects the structure of the algorithm. A_D did not vary as much as in the other communication intensive examples. Maximum logical delay is substantially higher, once again a reflection of the algorithm.

The CM-RC example indicates a good match between algorithm and architecture. A faster transport system will not benefit the performance of this type of application significantly. It is important for RC that the communication frequency is kept high enough to keep all processors busy, while at the same time not starving them for work or overrunning them with too much communication. The latter (extreme) case is not considered.

5.0.4 CP-Ring. As expected for a mostly computationally bound program, like CP-Ring, the normalized components show little change with dilation. User computation time makes up the majority of the overall execution time. Since the communication components demand little processing in such an application, clearly only slight benefit is derived from a faster transport system.

The speedup for this example is negligible. No matter how much the communication system is enhanced, this application stands little chance for improvement. Since the computation load is high, the speed of the processor or more parallelism is the logical choice of investigation for performance improvement.

5.0.5 CP-Mesh. Results from CP-Mesh mirror those found in CP-Ring. However, a few differences do occur but seem to have little impact on results. The number of message retransmissions remains fairly constant with dilation. CP-Mesh indicates that light but concentrated communication traffic affects the communication system adversely. Recall that each node in Mesh requires four messages per communication step. It is not evident in this case that performance degradation results.

5.0.6 CP-RC. Results from CP-RC differ significantly from CP-Ring and CP-Mesh. Although the computation-communication granularity is high (computation intensive) for CP-RC, a large percentage of its overall execution time is spent in logical communication wait states. For example, in CP-RC-1 logical delays make up 48% of the overall execution time. Logical delay is reduced only sightly in CP-RC-2, and physical and overhead delays are insignificant. As a result, this application shows little improvement with a faster transport system. Note that the number of total messages is small and the number of messages with logical delays is even smaller, although the wait duration for each is very large. In fact, for CP-RC-1 A_D is 2.8 seconds and the maximum logical delay is in excess of 14 seconds. This work starvation clearly indicates a load imbalance. The user can conclude from this information that too many nodes are idle too often; maybe the application should be redesigned.

Message retransmission is not a problem for CP-RC since communication traffic is minimal. For the example test set, retransmissions have been closely linked to communication congestion at each node. Retransmissions can be very costly or irrelevant to application performance. More importantly, architecture characteristics like this should be apparent to programmers. Measurement instrumentation provides a means to detect such idiosyncrasies. Modeling and simulation may have difficulties in doing the same, since the effects are highly non-linear.

5.1 Summary of Results

Figure 7 illustrates the speedup curves for our example applications. Communication intensive applications with synchronous communication (CM-Ring and Cm-Mesh) show good speedup with the time dilation technique. Even though the Ring and Mesh differ in program structure, their response to the faster transport system are basically the same. RC shows little speedup in either instance.

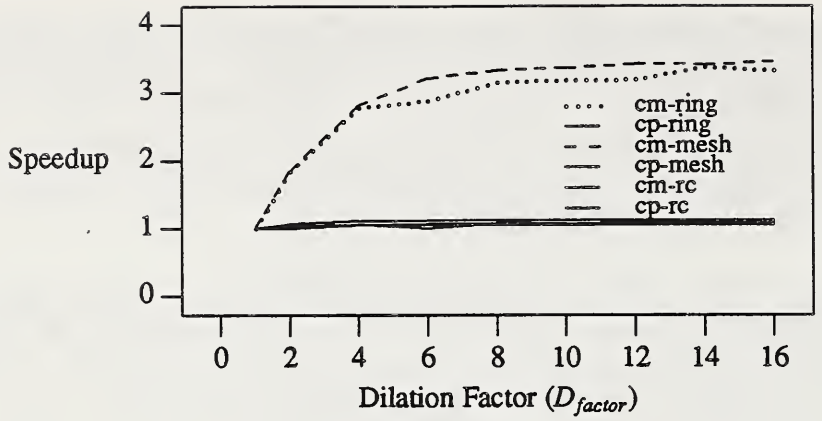

Figure 7. Speedup Curves for the Example Test Set.

A minor modification to a state occupancy, regardless of its resource usage magnitude, can significantly impact program behavior. The sensitivity slope for program elapsed time (E_D) versus physical delay (P_D) demonstrates this. The distinction of the logical communication structure of the algorithm (synchronous or asynchronous) can be important for determining the response to state occupancy modifications. Another important factor which contributes to the increased performance seen with the dilation experiments is the decreased frequency of message retransmissions as a faster physical transport system is emulated. The volume of traffic in the communication network has a direct relationship to retransmissions. A system that is relatively immune from this problem may react more conservatively to dilation. This highlights the importance of all communication components of the system.

	Application Paradigms	
	Ring, Mesh (Synchronous)	**RC (Asynchronous)**
Communication Intensive	- large logical delay - nice speedup - application benefits from faster transport	- small logical delay - little improvement - good match between algorithm-architecture - need faster cpu or more parallelism
Computation Intensive	- small logical delay - little improvement - good match between algorithm-architecture - need faster cpu or more parallelism	- large logical delay - little improvement - inefficient algorithm - consider algorithm redesign

Table II. Summary of Results.

Table II is an amalgam of results previously discussed. It highlights potential problem areas, as well as good architectural and algorithmic features. The table also suggests areas that should be considered for performance improvement. The results do not necessarily apply to all or any other hypercube applications. However, the results do demonstrate that this method of performance evaluation can extract revealing performance differences. Since the analysis is based upon system resources, which are invariant across all applications, it forms a reliable base for comparison. Thus the method is

fundamental for the study of all hypercube applications.

6. Summary and Conclusion

Time dilation provides an accurate method for investigating the performance of an application using a variety of physical transport speeds. Measuring sources of communication delay in a global domain reveals performance data that are important in the analysis of loosely-coupled machine applications. Together, time dilation and communication delay measurement provide an environment that offers insight for the development and analysis of concurrent algorithms and architectures. Logical and physical transport latencies indicate whether a poorly performing application needs a new algorithm (less logical delay) or a faster interconnection hardware (less physical delay). Other algorithmic and architectural characteristics are revealed with our performance evaluation measurements. These runtime statistics should be available to programmers so that necessary steps can be taken to improve their programs.

6.1 Acknowledgments

The original design sketch for time dilation was proposed by Gordon Lyon. He also read earlier versions of the text and suggested numerous details for improvement. John Antonishek implemented time dilation and instrumented the operating system so that event tracing could be performed.

7. References

[1] Antonishek, J. and Snelick, R. "Emulation through Time Dilation." Proceeding, Fifth Distributed Memory Computing Conference, DMCC5, Charleston, S.C., 1990, 8pp.

[2] Lyon, G. Design Factors for Parallel Processing Benchmarks. *Jour. of Theoretical Computer Science* April, 1989, 175-189.

[3] Lyon, G. and Snelick, R. Architecturally-Focused Benchmarks with a Communication Example. NISTIR 89-4053, March 1989, 38pp.

[4] Lyon, G. Capturing Some Practical Distributed-Memory Run-Time Statistics. NISTIR 4418 revised, December 1990, 12pp.

[5] Roberts, J., Antonishek, J. and Mink, A. *Hybrid Performance Measurement Instrumentation for Loosely-Coupled MIMD Architectures*, The Fourth Conference on Hypercubes, Concurrent Computers, and Applications, Monterey, CA, March 1989.

8. Appendix A

8.1 Performance Data for Ring

D	E_D	U_D	L_D	P_D	O_D	%LM	A_D	Max Log	TM	Retries
1	17.86	3.05	9.25	0.86	4.70	25	16.4	81.0	35808	482
2	9.71	3.05	3.83	0.43	2.40	43	3.9	121.9	35808	242
4	6.43	2.90	1.70	0.22	1.61	46	1.7	245.4	35808	58
6	6.20	2.89	1.64	0.14	1.53	45	1.6	331.4	35808	63
8	5.67	2.84	1.25	0.11	1.47	46	1.2	397.3	35808	39
10	5.62	2.84	1.24	0.09	1.45	45	1.2	471.4	35808	47
12	5.60	2.82	1.23	0.07	1.48	43	1.3	509.2	35808	41
14	5.29	2.79	0.98	0.06	1.46	44	1.0	689.4	35808	37
16	5.38	2.81	1.09	0.05	1.43	45	1.1	1064.7	35808	41
unit	s	s	s	s	s	%	ms	ms	count	count

Table A1. Performance Data for CM-Ring.

D	E_D	U_D	L_D	P_D	O_D	%LM	A_D	Max Log	TM	Retries
1	28.05	24.68	2.06	0.24	1.07	21	30.3	123.6	5088	385
2	26.25	24.62	0.94	0.12	0.57	25	11.9	81.5	5088	297
4	25.12	24.57	0.20	0.06	0.29	36	1.8	87.8	5088	78
6	25.01	24.56	0.16	0.04	0.25	40	1.3	80.8	5088	3
8	24.96	24.57	0.12	0.03	0.24	36	1.0	80.1	5088	9
10	24.99	24.57	0.16	0.02	0.24	38	1.3	289.3	5088	45
12	24.94	24.57	0.12	0.02	0.23	35	1.1	366.6	5088	22
14	24.96	24.56	0.16	0.02	0.22	43	1.2	566.8	5088	23
16	24.90	24.55	0.13	0.02	0.20	45	0.9	368.4	5088	3
unit	s	s	s	s	s	%	ms	ms	count	count

Table A2. Performance Data for CP-Ring.

8.2 Performance Data for Mesh

D	E_D	U_D	L_D	P_D	O_D	%LM	A_D	Max Log	TM	Retries
1	18.91	4.16	6.80	0.65	7.30	40	10.7	90.1	25600	862
2	10.23	4.10	3.16	0.33	2.64	40	4.9	81.1	25600	647
4	6.72	4.07	1.11	0.16	1.38	28	2.5	93.3	25600	462
6	5.88	3.99	0.30	0.11	1.48	13	1.4	119.6	25600	343
8	5.67	3.97	0.15	0.08	1.47	10	0.9	66.6	25600	298
10	5.62	3.96	0.14	0.07	1.45	7	1.2	503.9	25600	206
12	5.52	3.95	0.04	0.05	1.48	5	0.5	60.4	25600	231
14	5.54	3.96	0.04	0.05	1.49	6	0.5	54.1	25600	189
16	5.46	3.95	0.01	0.04	1.46	3	0.3	43.9	25600	157
unit	s	s	s	s	s	%	ms	ms	count	count

Table A3. Performance Data for CM-Mesh.

D	E_D	U_D	L_D	P_D	O_D	%LM	A_D	Max Log	TM	Retries
1	23.13	21.91	0.16	0.23	0.83	6	13.6	54.9	3200	167
2	22.22	21.85	0.07	0.11	0.19	3	10.1	70.3	3200	123
4	21.99	21.84	0.02	0.06	0.07	3	3.2	48.4	3200	184
6	21.91	21.83	0.01	0.04	0.03	2	2.3	73.5	3200	195
8	21.90	21.84	0.01	0.03	0.02	3	1.3	58.4	3200	144
10	21.88	21.84	0.00	0.02	0.01	3	0.7	45.5	3200	96
12	21.87	21.83	0.00	0.02	0.01	3	0.8	65.7	3200	149
14	21.87	21.83	0.00	0.02	0.01	3	0.6	46.9	3200	94
16	21.86	21.83	0.00	0.01	0.01	2	0.6	42.5	3200	112
unit	s	s	s	s	s	%	ms	ms	count	count

Table A4. Performance Data for CP-Mesh.

8.3 Performance Data for RC

D	E_D	U_D	L_D	P_D	O_D	%LM	A_D	Max Log	TM	Retries
1	24.85	21.65	1.61	0.16	1.43	3	75.0	405.4	13374	92
2	23.91	21.50	1.36	0.08	0.97	2	68.1	1280.0	13384	46
4	23.56	21.39	1.27	0.04	0.86	2	63.3	1635.2	13277	18
6	23.18	21.34	1.00	0.03	0.81	2	63.7	2795.9	13092	16
8	23.13	21.32	1.00	0.02	0.79	2	57.8	3261.3	13008	8
10	23.30	21.31	1.19	0.02	0.78	2	63.2	6267.9	13028	5
12	23.19	21.31	1.09	0.01	0.78	2	63.3	6402.5	13027	7
14	23.09	21.30	1.03	0.01	0.75	2	59.5	4994.5	12871	7
16	23.13	21.31	1.05	0.01	0.76	2	63.1	8127.3	13014	10
unit	s	s	s	s	s	%	ms	ms	count	count

Table A5. Performance Data for CM-RC.

D	E_D	U_D	L_D	P_D	O_D	%LM	A_D	Max Log	TM	Retries
1	31.30	16.11	15.16	0.00	0.03	36	2852.7	14,073.0	235	0
2	31.12	16.10	15.00	0.00	0.02	39	3334.4	20,645.7	187	0
4	29.47	16.10	13.36	0.00	0.01	36	2814.1	67,405.8	211	0
6	31.26	16.10	15.15	0.00	0.01	38	2491.2	111,404.2	194	0
8	29.02	16.10	12.92	0.00	0.00	34	3234.0	112,248.7	188	0
10	29.42	16.10	13.32	0.00	0.00	32	3580.7	168,348.6	186	0
12	28.57	16.10	12.46	0.00	0.00	33	3215.5	171,732.2	190	0
14	28.33	16.10	12.23	0.00	0.00	33	3106.0	196,953.2	189	0
16	28.33	16.10	12.22	0.00	0.00	33	3103.5	225,059.4	189	0
unit	s	s	s	s	s	%	ms	ms	count	count

Table A6. Performance Data for CP-RC.

NIST-114A
(REV. 3-89)

U.S. DEPARTMENT OF COMMERCE
NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY

BIBLIOGRAPHIC DATA SHEET

1. PUBLICATION OR REPORT NUMBER
NISTIR 4630

2. PERFORMING ORGANIZATION REPORT NUMBER

3. PUBLICATION DATE
JULY 1991

4. TITLE AND SUBTITLE

Performance Evaluation of Hypercube Applications: Using a Global Clock and Time Dilation

5. AUTHOR(S)

Robert D. Snelick

6. PERFORMING ORGANIZATION (IF JOINT OR OTHER THAN NIST, SEE INSTRUCTIONS)

U.S. DEPARTMENT OF COMMERCE
NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
GAITHERSBURG, MD 20899

7. CONTRACT/GRANT NUMBER

8. TYPE OF REPORT AND PERIOD COVERED

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS (STREET, CITY, STATE, ZIP)

Defense Advanced Research Projects Agency
1400 Wilson Boulevard
Arlington, VA 22209

10. SUPPLEMENTARY NOTES

[ ] DOCUMENT DESCRIBES A COMPUTER PROGRAM; SF-185, FIPS SOFTWARE SUMMARY, IS ATTACHED.

11. ABSTRACT (A 200-WORD OR LESS FACTUAL SUMMARY OF MOST SIGNIFICANT INFORMATION. IF DOCUMENT INCLUDES A SIGNIFICANT BIBLIOGRAPHY OR LITERATURE SURVEY, MENTION IT HERE.)

The speed of communication is very important in the performance of programs for loosely-coupled machines. A precursory study introduced an emulation technique called time dilation [1] that investigated the effects of communication speeds by varying the ratio of two parameters: communication time and computation time. The analysis of this technique was based on local measurements of a node's system resource utilization. A problem for loosely-coupled systems with local clocks is that observations cannot directly resolve communication delays into logical and physical transport contributions. This work separates the communication component of a program (via a global clock) into two states: logical and physical delays. True measurements calibrate the indirect dilation method for a sharper, quantitative interpretation, that it does not otherwise have.

Time dilation, coupled with low perturbing global measurement, provides an environment that offers insight for the development and analysis of concurrent algorithms and architectures. Measurement of communication service states indicates categorically sources of delay in the communication structure. The observables are applied to evaluate a set of example hypercube applications. Since the analysis is based upon system resources, which are invariant across all hypercube programs, it forms a reliable base for comparison. Results demonstrate that improvement to infrequent system states can cause significant changes in program behavior. This is most apparent for synchronous algorithms.

12. KEY WORDS (6 TO 12 ENTRIES; ALPHABETICAL ORDER; CAPITALIZE ONLY PROPER NAMES; AND SEPARATE KEY WORDS BY SEMICOLONS)

communication; emulation; global clock; hypercube; measurements; performance; resource usage; service states; time dilation

13. AVAILABILITY

[X] UNLIMITED

[ ] FOR OFFICIAL DISTRIBUTION. DO NOT RELEASE TO NATIONAL TECHNICAL INFORMATION SERVICE (NTIS).

[ ] ORDER FROM SUPERINTENDENT OF DOCUMENTS, U.S. GOVERNMENT PRINTING OFFICE, WASHINGTON, DC 20402.

[X] ORDER FROM NATIONAL TECHNICAL INFORMATION SERVICE (NTIS), SPRINGFIELD, VA 22161.

14. NUMBER OF PRINTED PAGES
26

15. PRICE
A03

ELECTRONIC FORM